Some Positive Results for Boundedness of Multiple Recursive Rules

Ke Wang

Department of Information Systems and Computer Sciences
National University of Singapore
Lower Kent Ridge Road, Singapore 0511

Abstract. Following results are sketched in this extended abstract: (1) Datalog recursive programs where each rule has at most one subgoal called *unit recursions* are shown to be bounded, with an effective construction of equivalent non-recursive programs. (2) A *generalized chain program*, which allow IDB predicates of arbitrary arity and remove the uniqueness condition of chain variables, is bounded if and only if it is a unit recursion. (3) The characterization of uniform unboundedness for linear sirups in [NS] is extended to a substantial superclass called class C^+. (4) Boundedness for class C^+ with multiple exit rules is decidable in polynomial space. (5) Predicate boundedness is decidable in doubly exponential time for a large class of Datalog programs that properly contains all connected monadic programs. (6) For binary linear programs, program boundedness is decidable if each recursive predicate is defined by at most one recursive rule; predicate boundedness is also decidable if each recursive predicate is mutually recursive with one another.

1 Introduction

This abstract presents some positive results of the boundedness problem for logic programs with multiple rules and multiple recursive predicates. The boundedness problem is to answer whether a given recursive program is equivalent to a non-recursive program, i.e., whether the program is *bounded*. Detecting bounded programs is a powerful optimization technique as a bounded program needs only a fixed number of iterations in evaluation or can simply be replaced by a non-recursive program. Unfortunately, this problem is undecidable in many cases, which include, among others, programs with a single recursive rule [Ab], linear programs with one binary IDB predicate [Va], and programs with two linear recursive rules and one initialization rule [Va]. Because of the inherent difficulty of boundedness problem, the positive results in earlier work [HKMV, Io, Na, NS, Va] have been obtained mainly for programs of a single recursive rule except for monadic programs [CGKV], some strongly restricted chain rules [AP, BKBR, Gu] that correspond naturally to productions of a context-free grammar, as well as typed rules with a single predicate (not only a single IDB predicate) [S]. There is a lack of positive results for more general rules.

The following are the contributions in this paper.

- (Section 3) Datalog programs in which each recursive rule has at most one subgoal are bounded. Such programs are called *unit recursions* in this paper. The result is also extended to a more general case, called *pseudo-unit recursions*, where each recursive rule has at most one recursive subgoal and the variables in all non-recursive subgoals occur in the recursive subgoal. A construction of a non-recursive program that is equivalent to a unit recursion (resp., pseudo-unit recursion) is presented. The constructed non-recursive program may have many rules, but each rule is very simple and the depth of the program is very small, a feature desirable for parallel evaluation.
- (Section 4) Reduction of boundedness to finiteness of CFL for "chain rules" [AP, BKBR, Gu] is extended to more general programs, called *generalized chain programs*. It is shown that a generalized chain program is bounded if and only if it is a unit recursion. In all "chain rules" studied previously in the literature, uniqueness of chain variables has been a crucial requirement for mapping rules to productions of a CFG. Our generalization is substantial in that IDB predicates of arbitrary arity are allowed and uniqueness of chain variables is no longer required.
- (Section 5) We extend the characterization of uniform unboundedness for linear sirups in [NS] to a superclass of the class C defined there, which we call class C^+. For a linear sirup in C^+, the restriction that no linking variables are mapped to persistent variables, which is a crucial requirement in [NS], is removed. All linear sirups efficiently identified as in C by methods in [NS] as well as more linear sirups can be efficiently identified as in C^+ by a method given in this paper.

We also extend the language (or automata) theoretic approach in [CGKV] for monadic programs to arbitrary Datalog programs. In particular,

- (Section 6) We show that boundedness (not just uniform boundedness) for linear sirups with a recursive rule in class C^+ and with multiple exit rules is decidable in polynomial space. Positive results of boundedness were obtained in [NS] only for a refinement of class C with one strongly restricted exit rule.
- (Section 7) We show that predicate boundedness is decidable in doubly exponential time for a large class of arbitrary Datalog programs that properly contains all connected monadic programs [CGKV].
- (Section 8) We show that program boundedness is decidable for binary linear programs in which each recursive predicate is defined by at most one recursive rule and that predicate boundedness is also decidable if each recursive predicate is mutually recursive with one another. These results generalize the decidability of boundedness for linear binary sirups in [Va].
- In the spirit of [CGKV], it follows immediately from our results that containment problem is decidable for programs considered in Sections 6,7 and 8.

Due to compactness of the presented materials, certain knowledge of work in [CGKV, Gu, Na, NS, Va] is helpful in reading the paper.

2 Preliminary

A program has an *IDB graph* in which nodes are IDB predicates of the program and there is a directed edge $< q, p >$ if q occurs in the body of a rule whose head predicate is p; we say that this rule *contributes* to this edge. An IDB predicate q is *useful* to an IDB predicate p if there is a (directed) path from q to p in the IDB graph; otherwise, q is *useless* to p. A *sirup* consists of one recursive rule and some number of exit rules. A *monadic* program is a program in which all IDB predicates have arity one.

A DB of a program P is a set of ground predicate instances, called *tuples*, for the predicates in P. An EDB is a DB in which the set of tuples for the IDB predicates in P is empty. For each DB or EDB I, $q_P^i(I)$ denotes the set of tuples for an IDB predicate q that can be derived by at most i applications of rules in P, and let $q_P^\infty(I) = \cup_{i \geq 0} q_P^i(I)$. A program P_1 is *contained* (resp., *uniformly contained*) in a program P_2 wrt q if $q_{P_1}^\infty(I) \subseteq q_{P_2}^\infty(I)$ for every EDB (resp., DB) I. P_1 is *equivalent* (resp., *uniformly equivalent*) to P_2 wrt q if P_1 is contained (resp., uniformly contained) in P_2 wrt q and vice versa. q is *bounded* (resp., *uniformly bounded*) in P if there exists some k, depending only on P, such that for every EDB (resp., DB) I, $q_P^\infty(I) = \cup_{i=0}^k q_P^i(I)$. This testing is called *predicate boundedness* problem. P is *bounded* (resp., *uniformly bounded*) if q is bounded (resp., uniformly bounded) in P for every IDB predicate q in P. This testing is called *program boundedness* problem. It is known that q is bounded (resp., uniformly bounded) in P if and only if P is equivalent (resp., uniformly equivalent) to a non-recursive program wrt q. Note that: (1) decidability of program boundedness does not necessarily imply decidability of predicate boundedness and (2) uniform boundedness implies boundedness.

Let q be an IDB predicate in a program P. A *partial q-expansion* is a conjunction of predicate instances that can be generated by some sequence of backward applications of rules in P beginning with an instance of q containing distinct distinguished variables (dv's). See [NS] for a detailed definition of backward applications of rules. A *q-expansion* is a partial q-expansion that contains only EDB predicates. The relation specified by a q-expansion $A_1, \ldots, A_n$ is

$$\{(v_1, \ldots, v_i) \mid (\exists w_1) \ldots (\exists w_j)(A_1 \wedge \ldots \wedge A_n)\},$$

where v's are dv's and w's are non-distinguished variables (ndv's, i.e., existential variables) introduced by backward applications. Given an EDB I, $q_P^\infty(I)$ is equivalent to the infinite union of the relations specified by all q-expansions. It was shown in [Na] that q is bounded in P if and only if for q-expansions $C_0, C_1, \ldots$ there is some $N \geq 1$ such that for every $n > N$, the relation specified by C_n is contained in C_m for some $m \leq N$. Note that C_n is contained in C_m if and only if there is a *containment mapping* from C_m to C_n [CM].

3 Boundedness of Unit and Pseudo-Unit Recursions

We first consider programs in which each recursive rule has only one subgoal. We show that such programs are always bounded by constructing a non-recursive

equivalent program.

3.1 Unit Recursions

Definition 1. *A recursive rule in a program is a* unit rule *if it has exactly one subgoal; otherwise, it is a* non-unit rule. *A program is a* unit recursion *if every recursive rule in it is a unit rule.*

Clearly, a program P is not a unit recursion if and only if the IDB graph of P has a cycle on which at least one edge is contributed by a non-unit rule in P. We will call such cycles *non-unit cycles.*

Example 1. Consider a program consisting of the following rules:

$$
\begin{array}{ll}
r_1: & p(x,y,z) :- q(x,y,z,z) \\
r_2: & r(z,x,y) :- p(x,y,z) \\
r_3: & q(x,x,y,z) :- r(x,y,z) \\
r_4: & q(w,x,y,z) :- e(w,x,y,z) \\
r_5: & v(x,y,z) :- r(x,y,w), e'(w,y,z) \\
r_6: & p(x,y,y) :- v(x,y,z), q(z,w,x,y) \\
r_7: & r(x,z,y) :- p(x,y,z), u(x,y)
\end{array}
$$

This program is not a unit recursion because non-unit rules r_5, r_6, r_7 contribute to edges of a cycle in the IDB graph. However, a program consisting only of the first five rules is a unit recursion, since in this case r_5 is not a recursive rule and all recursive rules r_1, r_2, r_3 are unit rules.

The next example illustrates some basic idea of constructing an equivalent non-recursive program for a unit recursion.

Example 2. Consider the unit recursion given by the first five rules $r_1, \ldots, r_5$ in Example 1. We want to find a non-recursive program equivalent to this program wrt $Q = \{q, v\}$. Suppose that the relation for q receives a "canonical" tuple t, i.e., a tuple consisting of distinct variables, through initialization rule r_4. Other tuples can be derived from t by applying recursive rules r_1, r_2, r_3. In applying rules, we treat variables in t as unknown values so that they can be equated or replaced with constants if necessary for applying a rule. For instance, to apply r_1 to t the last two variables in tuple t must be equated. Assume we have derived all tuples from t by applying rules r_1, r_2, r_3 in this manner. Let t' be any derived tuple for q (or r). The variables in t' must appear in the original tuple t because all rules are safe. Let t'' be obtained from t by equating or replacing (with constants) whatever variables that were equated or replaced in the derivation of t'. Then the derivation of t' (from the corresponding tuple in e) can be represented by a rule $q(t') :- e(t'')$ (or $r(t') :- e(t'')$). If we so construct a rule for each derived t', the recursive rules r_1, r_2, r_3 can be removed because each derivation has been represented by one of these new rules. As a result, the program becomes non-recursive and equivalent to the original program wrt $\{q, v\}$.

In the above example, the canonical tuple for q is transformed by a block of mutually recursive rules into tuples for predicates, q and r, that are useful for answering the query, q and v. In each transformation, we are interested in the mapping from the initial tuple (possibly with some variables equated or replaced by constants) to the final derived tuple, not the intermediate steps in the transformation. If each recursive rule is a unit rule, there are only a bound number of such mappings and each mapping can be represented by a non-recursive rule. In fact, we can effectively construct all non-recursive rules representing mappings.

Theorem 2. *Let P be a unit recursion and let Q be a set of query predicates. a non-recursive program equivalent to P wrt Q can be constructed effectively.*

Due to space limitation, the construction is omitted here.

3.2 Pseudo-Unit Recursions

Now we extend the algorithm in subsection 3.1 to more general programs.

Definition 3. *A recursive rule in a program is a* pseudo-unit rule *if it has the form*

$$p :- q, e_1, \ldots, e_k, \quad k \geq 0$$

where (a) q is mutually recursive with p and none of $e_1, \ldots, e_k$ is mutually recursive with p, (b) every variable that appears in some of $e_1, \ldots, e_k$ also appears in q. A program is a pseudo-unit recursion *if every recursive rule is a pseudo-unit rule.*

The attachment $e_1, \ldots, e_k$ in the above pseudo-unit rule can be considered as "conditions" on tuples for the subgoal q. By modifying the concept of mappings defined for unit recursions to account for such "conditions", we can effectively construct all non-recursive rules that represent mappings as in subsection 3.1. So we have

Theorem 4. Let P be a pseudo-unit recursion. For any non-empty set Q of query predicates, a non-recursive program equivalent to P wrt Q can be constructed effectively.

4 Generalized Chain Programs

In this section, we consider several classes of programs for which the condition of unit recursion is both necessary and sufficient for boundedness.

Definition 5. *A program P is a* generalized chain program *if for every predicate p there are two distinct positions h_p and t_p, called the* head *position and the* tail *position of p, such that every rule in P has the form (up to reordering of subgoals):*

$$q(\mathbf{x}) :- q_1(\mathbf{y}_1), \ldots, q_k(\mathbf{y}_k), \quad k \geq 1,$$

where (a) $\mathbf{x}[h_q] = \mathbf{y}_1[h_{q_1}]$, $\mathbf{x}[t_q] = \mathbf{y}_k[t_{q_k}]$, *and (b) for* $1 \leq i < k$, $\mathbf{y}_i[t_{q_i}] = \mathbf{y}_{i+1}[h_{q_{i+1}}]$ *and are (not necessarily distinct) ndv's. (We use* $\mathbf{u}[i]$ *to denote the ith argument of* $\mathbf{u}$*) A generalized chain program is* simple *if in each rule a ndv appears in at most one head position (or tail position). A program is* stationary *if every rule in the program has the form* $q(\mathbf{x}) :- q_1(\mathbf{x}), \ldots, q_k(\mathbf{x})$, *where* $\mathbf{x}$ *is a vector of distinct dv's.*

A stationary program is obviously bounded. Unlike all previously studied "chain rules" [AP, BKBR, Gu], we do not require that ndv's as chain variables be distinct. Even the most restricted class of simple generalized chain programs contains properly all chain programs in [AP, BKBR, Gu].

Theorem 6. (1) Every uniformly connected program in [Gu] that is not stationary is a simple generalized chain program. (2) Every general chained program in [Gu] is a simple generalized chain program. The converse is not true for each of (1) and (2).

Example 3. Consider the following simple generalized chain program:

$$q(x_1, x_2, x_3) :- q(u, x_2, u), q(x_1, u, x_3),$$

where $h_q = 2$ *and* $t_q = 3$. *This rule is uniformly connected but not general chained [Gu]. The following simple generalized chain program:*

$$q(x_1, x_2, x_3) :- q(x_1, u, z), q(z, x_2, x_3),$$

where $h_q = 1$ *and* $t_q = 3$, *is general chained but not uniformly connected [Gu].*

Rule bodies in a generalized chain program are considered naturally as strings of predicate instances in the order they are written, so are partial expansions generated by such programs. In the following, we reduce boundedness of some generalized chain programs to finiteness of CFL. We associate with each Datalog program P and each IDB predicate q a CFL $L(P, q)$ generated by a grammar CFG $\Gamma(P, q)$ given below: With each EDB predicate b we associate a terminal symbol t_b, and with each IDB predicate p we associate a non-terminal symbol v_p. The productions of $\Gamma(P, q)$ are obtained by replacing in each rule of P all occurrences of predicates by the corresponding grammar symbols, deleting all the variables in the rules, and turning $:-$ into $\rightarrow$. The starting symbol of $\Gamma(P, q)$ is v_q, the symbol associated with q. The *graph* of a CFG is a directed graph that contains all non-terminal symbols as nodes, and contains a directed edge $< A, B >$ whenever there is a production $A \rightarrow \alpha$ such that B is in α.

Theorem 7. Let P be a simple generalized chain program and q be an IDB predicate. q is bounded in P if and only if $L(P, q)$ is finite.

The proof is essentially the same as in [Gu], that is, the uniqueness of variables in head (resp., tails) positions implies that the mapping from a q-expansion C_l to a q-expansion C_m always maps the ith predicate instance in C_l to the ith predicate instance in C_m. In the following, we consider reduction of some non-simple subclasses. The first such subclass is based on the observation that the above uniqueness is only required for head (resp., tail) positions of instances of the *same* predicate, since containment mappings preserve predicates.

Let P be a generalized chain program. Given two predicates p and q, we say that p is *directly left* (resp., *directly right) dependent* on q if either $p = q$ or there is a rule in P such that p is the head predicate and q is the predicate of the first (resp., last) subgoal in the body. The *left* (resp., *right) dependency* is defined to be the transitive closure of the direct left (resp., right) dependency.

Definition 8. *Let P be a generalized chain program and r be a rule in P. r is* type 1 *wrt P if the following conditions hold: if a ndv u occurs in two tail positions t_p and t_q in the body of r, then p and q are not right dependent on a common predicate, and if a ndv u occurs in two head positions h_p and h_q in the body of r, then p and q are not left dependent on a common predicate. P is* type 1 *if every rule in P is type 1 wrt P.*

For a generalized chain program of type 1, it can be shown that in any expansion variables at the head position in two instances of the same predicate are pairwise distinct. So we have

Theorem 9. Let P be a type 1 generalized chain program and q be an IDB predicate. q is bounded if and only if $L(P, q)$ is finite.

Reduction to finiteness of a CFL also holds as long as in any expansion all predicate instances sharing variables at head (or tail) positions are no more than a fixed number of predicate instances apart. In the following, an IDB predicate is called *recursion-related* in a program if either it is recursive or it can be reached from a recursive predicate in the IDB graph of the program.

Definition 10. *Let P be a generalized chain program and r be a rule in P. r is* type 2 *wrt P if for every recursion-related predicate instance p in the body of r, subgoals on the left of p and subgoals on the right of p have disjoint variables at all head (or tail) positions. P is* type 2 *if every rule in P is type 2 wrt P.*

However, disjointness of variables is not required for non-head (or non-tail) positions on the two sides.

Theorem 11. Let P be a type 2 generalized chain program and q be an IDB predicate. q is bounded if and only if $L(P, q)$ is finite.

Since for each q-expansion there is a word in $L(P, q)$ of the same length, as a corollary of proofs of the above reduction, boundedness of the above programs in fact implies boundedness of length of expansions, as stated in the following corollary.

Corollary 12. Let P be a generalized chain program that is either simple, or type 1, or type 2. An IDB predicate q is bounded in P if and only if q-expansions in P have a bounded length.

But q-expansions have a bounded length if and only if all recursive rules defining predicates useful to q are unit rules (as defined in Section 3). So we have an efficient characterization of boundedness for the above programs.

Theorem 13. Let P be a generalized chain program that is either simple, or type 1, or type 2. Let q be an IDB predicate. If P has no useless predicates to q, then the following are equivalent. (1) $L(P, q)$ is finite. (2) q is bounded in P. (3) The IDB graph of P has no non-unit cycle. (4) P is a unit recursion.

Based on Theorem 13, we can show that boundedness and uniform boundedness coincide for each of the above three subclasses of generalized chain programs.

Theorem 14. Let P be a generalized chain program that is either simple, or type 1, or type 2. Let q be an IDB predicate. Assume that P contains no useless predicates to q. Then q is bounded in P if and only if q is uniformly bounded in P.

5 Extending the A/V Graph Approach

In [NS], uniform boundedness for a linear sirup in a class called C is characterized by absence of chain generating paths in the A/V graph of the linear sirup. Unfortunately, membership in C is difficult to test and only some sufficient conditions are given there. We now extend that characterization to a superclass of C, called C^+, so that more programs can be efficiently identified to suit the characterization. See [NS] for definitions of *A/V graphs, persistent variables, linking variables, chain generating paths.* Non-persistent variables are variables (dv's or ndv's) that are not persistent.

Definition 15. *Let r be the recursive rule of a linear sirup defining an IDB predicate q. A non-persistent variable V in r is called a* target link *if for any pair C_i and C_j of q-expansions with $i \neq j$, whenever C_i maps to C_j, no instance of V is mapped to a persistent variable. r is in class C^+ if either it has no chain generating paths or has a chain generating path on which all linking variables (i.e., ndv's on this path) are target links.*

Note that a linear sirup is in class C if and only if all linking variables are target links. But for class C^+, being target links is required only for the linking variables on a single chain generating path. The following generalizes results in [NS].

Theorem 16. (1) $C \subset C^+$. (2) For every $r \in C^+$, r is uniformly unbounded if and only if r has a chain generating path.

Finding all target links is not an easy task. The following theorem finds enough target links so that all programs identified as in class C by all lemmas in [NS] (i.e., Lemmas 4.5, 4.6, 4.7, and 4.8) as well as some other programs can be efficiently identified as in class C^+. We first define a relationship between two predicate instances.

Definition 17. *Let r be the recursive rule of a linear sirup. Let e and e' be two (not necessarily distinct) predicate instances in the body of r. $e \leq e'$ if all following conditions hold: (a) they are instances of the same predicate, (b) if position t in e has a persistent variable X, then position t in e' has X, (c) if position t in e has a target link, then position t in e' has a non-persistent variable, and (d) if positions t and s in e share a variable, then positions t and s in e' share a variable.*

Intuitively, $e \leq e'$ is a necessary (but not necessarily sufficient) condition for an instance produced by e to be mapped to an instance produced by e' in any containment mapping between expansions generated by the linear sirup.

Theorem 18. Let r be the recursive rule in a linear sirup and V be a variable in r. Then V is a target link if one of the following holds. (1) (Basis) V is a ndv appearing in a non-repeating EDB predicate in r. (2) V is a ndv appearing in an EDB predicate e in r such that, for every other EDB predicate e' in r with $e \leq e'$, if V appears in a position t in e then a non-persistent variable appears in position t in e'. (3) V is a non-persistent dv that can be reached from a target link in the A/V graph of r. (4) V is a ndv such that, for two identical copies B and B' of the set of EDB predicate instances in the body of r, V is always mapped to some non-persistent variable in every containment mapping from B to B'.

Observe that these rules identify as target links only variables in r whose instances are never mapped to persistent variables in all potential containment mappings.

Example 4. Consider the rule r

$$t(X, Y, Z) :- t(X, U, V), a(X, X), a(V, Z), e(U, Y)$$

There is a chain generating path that contains U as the only ndv. By Theorem 18(1), U is a target link, so r is in C^+ and is not uniformly bounded by Theorem 16. Observe that X is a persistent variable that occurs in some linking position of a, since V is a linking variable (on a different chain generating path). Thus [NS] can not identify r as a member in class C and therefore can not tell if r is uniformly bounded, even though r may be indeed in C.

For the rest of the paper, we extend the language-theoretic approach in [CGKV] for monadic programs to programs of arbitrary arity.

6 Decidable Boundedness of C^+ Class

Testing boundedness is harder than testing uniform boundedness. Decidability of boundedness was given in [NS] only for a refinement of class C with one strongly restricted exit rule. By extending the language-theoretic technique [CGKV] we can show that boundedness for the whole class C^+ with arbitrary exit rules is decidable, as stated in the following theorem.

Theorem 19. Boundedness for linear sirups with a recursive rule in C^+ and multiple exit rules is decidable in polynomial space.

Proof idea: We extend the language-theoretic characterization of unboundedness for monadic programs to programs in C^+. Assume P is a linear sirup with a recursive rule r in class C^+. If r has no chain generating path then P is bounded [NS]; otherwise, r has a chain generating path on which all linking variables are target links. By unfolding r a certain number of times, boundedness of P can be reduced to boundedness of a linear sirup in which the recursive rule has the form (Theorem 4.1, [HKMV])

$$r': \quad p(X_1,\ldots,X_k,Y_1,\ldots,Y_l) :- p(X_1,\ldots,X_k,Z_1,\ldots,Z_l), A$$

where X's, Y's, and Z's are vectors of distinct variables and they share no variables, A is a conjunction of EDB predicates. Moreover, $r \in C^+$ implies that $r' \in C^+$, and there must be some integer $1 \leq m \leq l$ such that position $k+m$ of p is on a chain generating path (in the A/V graph of r') on which all linking variables are target variables. Clearly, Y_m is connected to Z_m in G_A and Z_m is a target link, where $G_A = (V, E)$ is the *variable graph* of F [CGKV], such that V is the set of variables occurring in A and $< X, Y > \in E$ if X and Y occur in the same predicate instance in A. Without loss of generality, assume the exit rules have the same head as r' and have the bodies $B_1,\ldots,B_n$. By a reduction in [CGKV] (i.e., Proposition 5.4), no generality is lost by assuming that each graph G_{B_j} is connected. $B_1,\ldots,B_n$ are called *initialization bodies* and A is called a *recursive body*.

The following extends the language treatment in [CGKV] to the above linear sirup. Let A^i (resp., B^i_j) be a variant of A (resp., B_j), where all variables carry a superscript i and if $i > 1$ Y^i_j are replaced by variables Z^{i-1}_j, $1 \leq j \leq l$. Initially, view A^1 (resp., $B^1_{i_1}$) as a conjunctive query where all variables except for dv's are existentially quantified and this one body is the *leaf*. Inductively, suppose that C is a conjunctive query, A^k is a recursive body that is the leaf of C, and $B_{i_{k+1}}$ is an initialization body. Note that A^k contains variables $Z^k_1,\ldots,Z^k_l$, which are also in A^{k+1} (resp., $B^{k+1}_{i_{k+1}}$). Then C, A^{k+1} (resp., $C, B^{k+1}_{i_{k+1}}$) can be viewed as a conjunctive query C', where all variables except for dv's are existentially quantified and where A^{k+1} (resp., $B^{k+1}_{i_{k+1}}$) is the leaf of C'. Let $\Sigma_r = \{a\}$ be the *recursive alphabet* and $\Sigma_i = \{b_1,\ldots,b_n\}$ be the *initialization alphabet*. It is important to see that each conjunctive query is uniquely determined by a word in $(\Sigma_r \cup \Sigma_i)^*$. We can show that the language-theoretic characterization of unboundedness for connected monadic programs, i.e., Proposition 3.2 in [CGKV],

is still valid for the above linear sirup. The key argument is that if there is a containment mapping from a p-expansion C_i to a p-expansion C_j, then there is a containment mapping from C_i to the prefix of C_j that contains no more than c bodies, for some fixed integer c. In particular, the position $k+m$ of p plays the role of the position of an unary recursive predicate in a monadic program, in that all variables Z_m^i are connected by EDB predicates and none of them is mapped to a persistent variable (because Z_m is a target link). The presence of variables in other positions of p does not affect the above argument. The rest of treatment is then a copy of [CGKV], because it depends only on language features that make no difference in our case.

7 Persistence-free and Connected Datalog Programs

We show that boundedness is decidable for a large class of Datalog programs that are not necessarily sirups. The general idea is to prevent "linking variables" from being mapped to "persistent variables". First we need these terms for general rules.

Definition 20. *Let P be a Datalog program, q an IDB predicate, and t a position of q. The position q^t is* persistent *wrt an IDB predicate p in P if for every $k > 0$ there is a partial p-expansion in which an instance of q contains a variable V at position t and the instance is at least k predicate instances away from the very first predicate containing V; otherwise, q^t is* persistence-free *wrt p in P. An IDB predicate p is* persistence-free *in P if q^t is persistence-free wrt p for every IDB predicate q and every position t of q.*

We create a directed graph for testing existence of persistence: G_{per} has a node p^t for each recursive predicate p and each position t of p; G_{per} has an (directed) edge from p^t to q^s if and only if there is a recursive rule with a p instance in the head and a q instance in the body such that the variable in position t in the head appears in position s in that q instance in the body.

Proposition 21. q^t is persistence-free wrt p if and only if no node of form p^s reaches a cycle containing node q^t in G_{per}.

The following definition generalizes the connectivity defined for monadic programs [CGKV] to Datalog programs.

Definition 22. *An IDB predicate p is* connected *in P if there is a choice of a position for each IDB predicate useful to p such that for each rule $q(X) :- A, q_1(Y_1), \ldots, q_n(Y_n)$ ($n \geq 0$), G_A is connected and $X, Y_1, \ldots, Y_n$ are in G_A, where q is useful to p, A is the conjunction of the EDB predicates in the rule, and $X, Y_1, \ldots, Y_n$ are variables in the chosen positions of predicates $q, q_1, \ldots, q_n$, respectively. (The rule is an exit rule when $n = 0$) The chosen positions, if they exist, are called* linking positions.

It is easy to see that every IDB predicate of a connected monadic program [CGKV] is persistence-free and connected (We need only consider monadic programs where the variable in the head does not appear in any IDB predicate in the body, as the general case can be reduced to this case [CGKV]). An efficient algorithm for testing connectivity and finding linking positions will be given in the full paper.

Theorem 23. Predicate boundedness for predicates that are both persistence-free and connected is decidable in doubly exponential time.

Proof idea: The idea is the same as Theorem 19, i.e., simulating a monadic program. Linking positions plays essentially the role of the single position of unary IDB predicates in a monadic program. Unlike Theorem 19, we have to use the general argument based on the tree language and tree automata [CGKV]. The time is doubly exponential because predicate boundedness for monadic programs is doubly exponential [CGKV].

However, the reduction of boundedness of unconnected programs to boundedness of connected ones for monadic programs in [CGKV] does not apply here, because the connectivity used in that reduction is weaker than the connectivity used here, although the two notions coincide for monadic programs.

8 1-branching Binary Linear Programs

It was previously known that boundedness is decidable for linear binary sirups but is undecidable for multiple recursive rules even with a single IDB predicate [Va]. We now show some positive results for linear binary programs, not necessarily sirups, where each predicate is defined by at most one recursive rule. First, we extend the decidability for binary linear sirups in [Va] to a slightly general version. In a sirup in [Va], all non-recursive predicates in the body of the recursive rule must be EDB predicates. A *generalized linear sirup* has a recursive rule of the form $p :- A, p$ (arguments are omitted here) and one or more non-recursive rules, where A is a conjunction of non-recursive predicates. In other words, non-recursive IDB predicates are allowed in the body of a generalized sirup. This generalization appears to be nontrivial because removing such non-recursive IDB predicates simply by unfolding them using non-recursive rules may result in more than one recursive rule, therefore, no longer a sirup. However, no generality is lost by assuming that each non-recursive rule has only EDB predicates in the body. We assume a generalized binary linear sirup has a recursive rule of the form $p(X, Y) :- A, p(U, V)$, for variables X, Y, U, V. Consider the following cases.

Case 1: X, Y, U, V are all distinct, and each of X and Y is connected to some of U and V in graph G_A and vice versa. By unfolding the recursive rule at most once, we can assume that X is always connected to U in G_A. We may also assume that for each exit rule with a body B, G_B is connected, since the general case can be reduced to this case by a reduction in [CGKV]. (Note that this is so only for exit rules) Now we remove all non-recursive IDB subgoals in

A by unfolding them using non-recursive rules. In each resulting recursive rule with the set A_i of EDB predicates, X is still connected to U in G_{A_i}. Therefore the first position of p can be chosen as the linking position. Now the predicate p is persistence-free and connected in the program under consideration. Then from Theorem 23, the boundedness is decidable.

Other cases: As in [Va], all other cases can be proved to be either bounded or reducible to monadic programs with a single recursive rule. It is important to see that those reductions do not depend on the absence of non-recursive rules. So we have

Theorem 24. Boundedness for generalized binary linear sirups is decidable.

Now we consider more general rules defined below.

Definition 25. *A program P is* 1-branching *if every recursive predicate is defined by at most one recursive rule.*

A 1-branching program may have more than one recursive predicate. In the following, we show that program boundedness is decidable for binary linear 1-branching programs. We first show that program boundedness is reduced to programs with at most one mutual recursion.

Let P be an arbitrary Datalog program. Let $I_1, \ldots, I_k$ be a partial ordering of the scc's of IDB graph of P such that no predicates in I_i depend on predicates in I_j for $j > i$. Let R_i be the set of rules in P that define predicates in I_i. $\{R_1, \ldots, R_k\}$ is a partition of rules in P.

Theorem 26. (1) P is bounded if and only if for $i = 1, \ldots, k$ in order, $R^n_{i-1} \cup R_i$ is bounded, where R^n_{i-1} is a non-recursive program equivalent to $R_1 \cup \ldots \cup R_{i-1}$, and $R^n_0 = \emptyset$. (2) P is uniformly bounded if and only if for every R_i, the set of recursive rules in R_i is uniformly bounded.

Corollary 27. (1) Program boundedness for 1-branching linear programs is reducible to boundedness for generalized linear sirups of the same arity. (2) Uniform program boundedness for 1-branching linear programs is reducible to uniform boundedness for linear sirups of the same arity. (3) Predicate boundedness and program boundedness coincide for 1-branching linear programs in which each recursive predicate is mutually recursive with one another.

Proof idea: Consider a 1-branching linear program consisting of three recursive rules $p :- A, q$, $\quad q :- B, r$, and $r :- C, p$, where A, B, C are conjunctions of non-recursive predicates. By unfolding these recursive rules, p can be defined by a linear recursive rule of form $p :- A, B, C, p$ and some number of non-recursive rules. Once p is so defined, q and r can be defined by p and other predicates non-recursively. As a result, we need only deal with a generalized linear sirup. This reduction also holds in general in light of Theorem 26.

From Corollary 27 and Theorem 24, we have

Theorem 28. (1) Program boundedness is decidable for 1-branching binary linear programs. (2) Predicate boundedness is decidable for 1-branching binary linear programs in which each recursive predicate is mutually recursive with one another.

Using the reduction of Corollary 27 and some known decidability for linear sirups in [Na, NS, HKMV], boundedness and uniform boundedness of many 1-branching linear programs of arbitrary arity can be shown to be decidable. Finally, in the spirit of [CGKV] (i.e., Proposition 7.1), we can show

Theorem 29. (1) Predicate containment is decidable in polynomial space for linear sirups in C^+ (with multiple exit rules). (2) Predicate containment is decidable in doubly exponential time for persistence-free and connected predicates. (3) Predicate containment is decidable for 1-branching binary linear programs in which each recursive predicate is mutually recursive with one another.

References

[Ab] Abiteboul, S.: Boundedness is undecidable for Datalog programs with a single recursive rules. IPL 32 (1989), pp. 281-287

[AP] Afrati, F., Papadimitriou, C.H.: The parallel complexity of simple chain queries. ACM PODS, 1987, pp. 210-213

[BKBR] Beeri, C., Kanellakis, P.C., Bancilhon, F., Ramakrishnan, R.: Bounds on the propagation of selection into logic programs. ACM PODS, 1987, pp. 214-226

[CGKV] Cosmadakis, S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimizations for datalog logic programs. ACM Symp. on Theory of Computing, 1988, pp. 477-490

[CM] Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. ACM Symp. on Theory of Computing, 1977, pp. 77-90

[GMSV] Gaifman, H., Mairson, H., Sagiv, Y., Vardi, M.Y.: Undecidable optimization problems for database logic programs. Proc. of 2nd IEEE Symposium on Logic in Computer Science, 1987, pp. 106-115

[Gu] Guessarian, I.: Deciding boundedness for uniformly connected Datalog programs. Lecture Notes in Computer Science 470, ICDT 1990, pp. 395-405

[HKMV] Hillebrand, G.G., Kanellakis, P.C., Mairson, H.G., Vardi, M.Y.: Tools for datalog boundedness. ACM PODS, 1991, pp. 1-12

[Io] Ioanidis, Y.E.: A time bound on the materialization of some recursively defined views. VLDB, 1985, pp. 219-226

[NS] Naughton, J., Sagiv, Y.: A decidable class of bounded recursions. ACM PODS, 1986, pp. 227-236

[Na] Naughton, J.: Data independent recursion in deductive databases. JCSS 38 (1989), pp. 259-289

[S] Sagiv, Y.: On computing restricted projections of representative instances. ACM PODS, 1985, pp. 171-180

[Va] Vardi, M.Y.: Decidability and undecidability results for boundedness of linear recursive queries. ACM PODS, 1988, pp. 341-351

This article was processed using the LaTeX macro package with LLNCS style